Description

Method and devices for encoding/decoding structured documents, especially XML documents

The invention relates to methods or devices for encoding structured documents, especially XML documents, in which a bit stream is generated from a structured document as a function of a schema, and a method or device for decoding, in which a structured document is generated from a bit stream as a function of a schema.

In the context of the work on the MPEG-7 standard, a method for binary encoding of XML data was developed which is referred to as the BiM method in the following and which is known, for example, from the publication ISO/IEC FDIS 15938-1:2001(E), "Information Technology - Multimedia Content Description Interface - Part 1: Systems". This method uses XML schema definitions which are available at the encoder and the decoder, such as the MPEG-7 schema, in order to generate the codes for the individual data elements of the XML description. A precondition for this method is that the same schema definitions are available at least in part to the encoder and the decoder. This can be ensured, for example, if a standardized XML schema is permanently installed in the decoder. Moreover, the possibility exists of transferring the schema to the decoder separately or in addition to the actual document. The transmission of the schema from the encoder to the decoder can be carried out in textual form, where standard text compression, such as ZIP, can be applied.

The object underlying the invention then consists in specifying methods or devices in such a way that the transmission of the schema is carried out especially efficiently and the dataset transmitted and the computing performance at the decoder, which is needed for generating the code tables from the schema, is

reduced. Moreover, the consistency of a schema which has not been transmitted in full should be ensured.

According to the invention, this object is achieved by the features of Claim 1 with regard to the encoding method, by the features of Claim 7 with regard to the decoding method, by the features of Claim 14 with regard to the encoding device and by the features of Claim 15 with regard to the decoding device.

The further claims relate to advantageous embodiments of the methods or devices according to the invention.

The invention essentially consists in producing a bit stream or part of a bit stream from a schema as a function of a metaschema with the aid of an encoding method, whereby at least one of the following optimization processes is carried out:
- separation of anonymous types from element declarations and attribute declarations and encoding as own type, the type definition thereof being instantiated in the schema definition as a top level element,
- normalization of the syntax trees on the encoder side,
- replacement of the character strings of type names,
- transmission of information for the inheritance tree.
The decoding takes said optimization processes into account and conversely produces a schema from the bit stream.

In the following, the invention is explained on the basis of exemplary embodiments shown in the drawings. In this respect,

Figure 1 shows a schematic diagram to explain the encoding/decoding according to the invention,

Figure 2 shows a diagram to explain the details of a preferred embodiment of the invention,

Figure 3 shows a diagram to explain the details of a further preferred embodiment of the invention and

Figure 4 shows a schematic diagram of a preferred embodiment of a decoder according to the invention.

Since XML schemas for their part comprise XML documents based on a standardized syntax definition, specifically what is referred to as a "schema for schemas" (W3C specification), which virtually represents a metaschema, a schema can also be encoded and transmitted with the aid of the BiM method referred to above.

Figure 1 shows an arrangement in which, in a first step, part of a bit stream or a bit stream BS1 is produced from an XML schema XMLS as a function of a metaschema SS with the aid of a BiM encoding method BiM-E and in which, in a second step, a further part of the bit stream or a bit stream BS2 is produced from an XML document XML as a function of the schema XMLS with the aid of the same BiM encoding method BiM-E and also in the opposite direction an XML schema and the XML document are recovered from the two parts of the bit stream or from the bit streams BS1 and BS 2 with the aid of a BiM decoding method BiM-D.

In a first preferred embodiment of the invention, the separation of what are referred to as "anonymous types" from the element or attribute declaration is performed.

The transmission of an XML document is effected "depth first" in the case of the BiM method, but the operation of schema compilation at the decoder demands a "breadth first" structure, where these expressions are explained in detail on the Internet page http://www.generation5.org/simple_search.shtml, for example. In the case of groups such as sequence or choice groups, this can be compensated for by means of a small buffer memory on the decoder side, but in the case of the "anonymous types", which can define the type of an individual element or attribute, the complexity justifies a restructuring process on

the encoder side: the anonymous type definitions, designated by AT0 in the following example, are taken out of the element declaration for the element "CurriculumVitae" and given a name and/or code which is used for referencing purposes in the case of the corresponding element.
By way of advantage, this reduces the depth of the hierarchy of the types transmitted, with the result that the compilation of the schema on the decoder side is simplified.

Example:

Schema prior to restructuring

```
<complexType name="PersonDescriptor">
   <element name="CurriculumVitae">
      <complexType>
         <element name="name" type="string"/>
         <element name="birthday" type="date"/>
            ...
      </complexType>
   </element>
   <element name="profession" type="profTp" />
</complexType>
```

Schema following restructuring

```
<complexType name="PersonDescriptor">
   <element name="CurriculumVitae" type="AT0"/>
   <element name="profession" type="profTp" />
</complexType>
```

```
<complexType name="AT0">
   <element name="name" type="string"/>
   <element name="birthday" type="date"/>
            ...
</complexType>
```

In a second preferred embodiment of the invention, the normalization of the syntax trees, as specified in BiM, is carried out on the encoder side.

In the BiM method, what are referred to as "Finite State Automatons", which are used for decoding the bit stream, are produced from syntax trees which map the structure of the XML schema. In order to enhance encoding efficiency, these syntax trees do not correspond 1:1 to the textual XML definitions; instead, normalizations are performed. Three different cases can occur in this respect:

1. Simplification of a group which contains only one element: the group is dissolved and the contained element is put into the content model at the level of the dissolved group, where the attributes minOccurs and maxOccurs of the element are replaced by the product of the corresponding attributes of the dissolved group and the element prior to the regrouping.

2. Simplification of a choice group containing an element with the attribute value minOccurs=0:
the attribute "minOccurs" of the choice group is set to 0 irrespective of the previous value, while the element which had an attribute value minOccurs=0 is assigned an attribute value minOccurs=1.

3. Simplification of nested choice groups:
if a choice group contains another choice group which contains the attribute values minOccurs=maxOccurs=1, that choice group is dissolved and the contents are incorporated directly into the superordinate choice group.

In the case of the transmission of the schema, these simplifications should already be performed at the encoder since the syntax tree transformations influence the allocation of the normative codes and the compilation of the schema is simplified on the decoder side if the content model can be taken over directly.

In this case, the advantages lie in the fact that this also relieves the burden on the decoder and the content model can be fed to the schema compiler directly as it is created in the type decoding.

In a third preferred embodiment of the invention, the replacement of the character strings of type names is carried out, as shown in Figure 2.

In the "name" and "base" attributes of a type definition, and also in the case of the "type" attribute of an element declaration or attribute declaration, the same type names occur frequently in the schema, which would be transmitted multiple times as character strings. In the case of type name encoding, therefore, it is advantageous to encode only a number in place of the name, and separately from this a table which links the numbers back to the original names. A suitable number comprises the type number, which the inheritance tree of the master type explained in further detail below allocates to all complexTypes.

The same also applies to the "name" attribute of global element declarations and their references in "ref" attributes, and to

the names of substitution groups in the "substitutionGroup" attribute. In these cases, the schema branch code SBC of the global elements can be used, for example.

This allows savings in data volume since a repeated reference to the same type name can be represented in more compact form and the type allocation table can be compressed better with a standard compression tool since the type names do not occur distributed throughout the bit stream, but in compact form in a connected area in the bit stream.

In an advantageous embodiment, a list comprising the type names or element names or names of substitution groups is encoded. Instead of explicitly allocating numbers to the names, the position of a name in the list is used as a number in this embodiment. This is advantageous since numbers no longer have to be encoded in the list and therefore more efficient transmission is ensured.

In a fourth preferred embodiment of the invention, the transmission of information for the inheritance tree is effected.

Each type definition contains, in what is referred to as the "base" attribute, if it is present, information as to which type it has been inherited from. Collecting all this information for a schema results in a tree structure, referred to as the inheritance tree. The inheritance tree is used in the case of the BiM encoding method to transfer the new type of the element in the event of a type conversion (type-cast). In this respect, the code allocated to all types inherited from the base type, that is to say what is referred to as the type code, and also the length of this code, is critical for correct decoding. The length is given by the overall number of all types in the inheritance tree under the base type. If the schema has been transmitted in full, both the codes and also the code length can be determined unambiguously on the decoder

side. If the schema is not complete on the decoder side, however, additional information still has to be transmitted in order to assign type codes to types which have already been transmitted.

Each transmitted type has the number of the type code with reference to the master type in the name field. This allows the type code of the derived types to be determined by means of simple difference formation. Still missing is the information about the power of the sub-tree defined by the transmitted types, and therefore the length of the type codes of the types derived from these transmitted types. This length can be transmitted with the aid of a few bits in a variable length code.

Figure 3 shows an inheritance tree of a schema with the type A, from which further types are derived, by way of example. This type is given the type code 134, for example, with reference to the master type "anyType". Derived from type A are the types AA, AB and AC, the type codes of which are specified with reference to the master type. In order to determine the type code with reference to the base type A, it is sufficient to subtract the type code of the base type and one from the type code of the desired type:

$$TC_{type} = TC_{type\ with\ ref.\ to\ master\ type} - TC_{base\ type\ with\ ref\ to\ master\ type} - 1$$

The missing information about the length of the type code can be integrated best in the reference table as an additional number.

In order to be able to compress the information in the type allocation table with a standard compression tool, it is advisable to store it aligned to whole bytes (byte-aligned). The first number comprises a vluimsbf5 number which encodes the number of lines in the table, followed by a vluimsbf5 number which encodes the number of bits for the type code, and a

further vluimsbf5 number which represents the type code with reference to the master type itself. Filler bits or stuffing bits follow in order to achieve the alignment to byte boundaries.

Format of the type allocation table			
Vuimsbf5	**Vuimsbf5**	**Bits**	**Character string**
Number of lines			
Length of type code 1	Type code 1	0-7 filler bits	Name of type 1
Length of type code 2	Type code 2	0-7 filler bits	Name of type 2
...	...	...	...

The transmission of a type allocation table makes it possible to correctly decode any type codes present in an encoded document even if the underlying schema has not, or not yet, been transmitted and/or decoded in full.

Correspondingly, the global SBC must be transferred with global elements, and the substitution code in the case of elements belonging to a substitution group, where one global SBC length and, with the header element of the substitution group, the length of the respective substitution code are transferred in advance for all global elements.

Any combination of the features represented in the individual embodiments is possible in the encoding and can also be used in corresponding fashion in the decoding.

The BiM method requires that the XML schema is compiled in a format which permits the stipulation of the length of the code words and the choice of the data elements by the values of the codes. There are several possibilities for this. The MPEG-7 standard (ISO/IEC 15938-1:2001 Part 1: Systems or ISO/IEC 15938-6:2001 Part 6: Reference software) proposes a model which uses finite state automatons for the decoding of the useful information or payload, and code tables which are generated from the schema for the decoding of a context path.

In a preferred embodiment of the decoder according to the invention shown in Figure 4, the decoding operation is described with the aid of a byte code model, where the schema structure is translated into a system of interlinked states which are processed by a byte code interpreter BCI, where a bit stream BS received from the encoder contains the information about the subsequent state to be chosen. In contrast to the model proposed in the MPEG-7 standard, the byte code model is created in such a way that both a bit stream representing a payload and also a bit stream representing a context path can be decoded. There is therefore no requirement to hold the same information contained in the schema twice at the decoder for the different encoding methods. The BCI interpreter reads the information from the incoming bit stream which encodes an XML document or an XML schema in the BiM format. This information allows a choice from among the subsequent states of the current state which is stored in the byte code. The subsequent states are created permanently as pointers P within the byte code. A path, payload or byte code is output depending on the configuration.

The decoding of a schema can also be implemented efficiently in the byte code model with the aid of the modifications proposed above. In this case, no payload or path is output; instead, byte code is produced directly which can be used by the byte code interpreter for the decoding of the corresponding types.

The byte code is made up of structural elements or the states. The states are of different types, identified with the aid of the header bit field of the state. The states contain different information fields depending on the type, which are read and, depending on the configuration (payload / context path) and current state, analyzed by the byte code interpreter.

Several variants are conceivable for the types of states which represent the schema information. The essential factor is that all the syntax elements of an XML schema can be reproduced by the states of the byte code model, and that all the information necessary for efficient decoding of the two algorithms defined in the MPEG-7 standard (context path / payload) is made available in the states.

A possible structure of the byte code is outlined briefly in the following.

Types of state: Overview

1. Header state of a complexType

The header state of a type forms the starting point in the decoding of a complexType. It contains the name of the type (if it does not constitute an anonymous type) and also information concerning inheritance of the type (pointer to base state) and also polymorphism.

A specific factor for payload encoding comprises a pointer to a list of the attributes of the type. A specific factor for context path encoding comprises fields with the number of child elements for the context and operand tree branch code tables. The last information field comprises a pointer to the subsequent state, i.e. the first state which represents the content of the complexType (an element state or a choice state, for example).

Graphical representation of a header state:

Header bit field
Pointer to string with name
Pointer to header state for base type
Pointer to inheritance tree
Number of children for context TBC
Number of children for operand TBC
Pointer to subsequent state

2. Choice state

A choice state reproduces a choice group of the XML schema. The choice state essentially contains a list of pointers with possible subsequent states. In order to stipulate the state actually chosen, the bit stream has to be read during the decoding of a payload. There are two variants of the choice state: a start state which branches into the different possible subsequent states, and also an end state which summarizes the choice again.

3. Element state

The element state reproduces an element declaration in a complexType of a schema. It contains a pointer to a character string with the name of the element, and also a pointer to the header state of the type. Furthermore, information may be present about the length of the position code (for path decoding only) and for substitution groups.

4. Attribute state

An attribute state reproduces an attribute declaration of a schema. It contains a pointer to the name of the attribute, and also a pointer to the header state of the simpleType of the attribute.

5. Occurrence state
An occurrence state reproduces the minOccurs and maxOccurs attributes which can occur in the case of an XML schema, e.g. in the case of an element or a group (choice, sequence, etc.). It contains a pointer to the subsequent state if a further instance of the element or group occurs, and also a pointer to the subsequent state if the last instance of the group has been encoded. Since the possibility exists of an element containing itself in the case of XML schemas (the element itself occurs again in the complexType definition of the element, or in an even deeper nesting), an occurrence state can also be active more than once at the same time. A pointer to a stack within the occurrence state is therefore required which secures the current state of each active instance of the occurrence state.

6. End state of a type
The end state of a type contains a list of pointers with all the attributes of that type. It is required in the decoding of a path since all the attributes are put in at the end of the table in the tree branch code table. Upon reaching an end state, the byte code interpreter branches hierarchically into the element which has called that type. The corresponding information about the calling element has to be stored in the working memory of the byte code interpreter.

7. Header state of a simpleType
This state controls the decoding of content, i.e. it contains a pointer to a codec which can specifically read and decode data of the relevant type from the bit stream. The type of the codec is specified in an information field.

The essential advantages of the byte code model compared to the status of the MPEG-7 reference software comprise:

1. The schema information is represented only once at the decoder for both encoding methods (context path / payload). The

major part of the information in the byte code states is relevant for both methods. A smaller part is specific for one of the two methods in each case. The representation of the schema information at the decoder is therefore very compact.

2. The byte code model makes a well-defined data format available for schema information which is also suitable for precompiling and saving, for example (in place of the XML schema as text).

3. The execution of the byte code by a standard processor can be carried out very rapidly since the byte code model prepares the decoding operation very effectively. All the information is available directly in the state via pointers and does not need to be partly searched for in lists first (as in ISO/IEC 15938-6, Part 6: Reference software).

A corresponding encoder can be implemented in the same manner, where it is inverted in the manner that the states are controlled by the textual representation of the structured document and the state transitions generate the binary representation.